A randomized block sampling approach to canonical polyadic decomposition of large-scale tensors

Nico Vervliet, *Student Member, IEEE,* and Lieven De Lathauwer, *Fellow, IEEE*

***Abstract*—For the analysis of large-scale datasets one often assumes simple structures. In the case of tensors, a decomposition in a sum of rank-1 terms provides a compact and informative model. Finding this decomposition is intrinsically more difficult than its matrix counterpart. Moreover, for large-scale tensors, computational difficulties arise due to the curse of dimensionality. The randomized block sampling canonical polyadic decomposition method presented here combines increasingly popular ideas from randomization and stochastic optimization to tackle the computational problems. Instead of decomposing the full tensor at once, updates are computed from small random block samples. Using step size restriction the decomposition can be found up to near optimal accuracy, while reducing the computation time and number of data accesses significantly. The scalability is illustrated by the decomposition of a synthetic $8\,\mathrm{TB}$ tensor and a real life $12.5\,\mathrm{GB}$ tensor in a few minutes on a standard laptop.**

***Index Terms*—Tensor decomposition, canonical polyadic decomposition, CANDECOMP/PARAFAC, randomized algorithms, block sampling, big data, blind source separation**

I. Introduction

With datasets growing in size and dimensions faster than ever, more efficient algorithms to analyze them are needed. Many datasets can be represented as multiway arrays of numerical values. These so-called tensors can be compressed or analyzed using a variety of decompositions such as a low multilinear rank approximation, a block term decomposition or tensor trains (see e.g. [1], [2]). In this paper we focus on the decomposition in rank-1 terms. A rank-1 tensor of order N is defined as the outer product, denoted by $\otimes$, of N vectors $\mathbf{a}^{(n)}$. The polyadic decomposition (PD) writes a tensor as a sum of R rank-1 terms:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{a}_r^{(1)} \otimes \cdots \otimes \mathbf{a}_r^{(N)} = \left[\!\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\!\right], \quad (1)$$

in which the factor matrices $\mathbf{A}^{(n)}$ have the R factor vectors $\mathbf{a}_r^{(n)}$ as columns. If R is the minimum number for which the equality holds, R is the rank of the tensor, and the polyadic decomposition (1) is called canonical (CPD). Up to trivial scaling and permutation indeterminacies, a CPD is unique under mild conditions [3]–[5], which has led to countless applications, e.g., in factor analysis [6], [7] and blind signal separation [1], [8], [9]. Interested readers are kindly referred to [1], [2] for a general background.

Since the introduction of the (C)PD, many algorithms have been developed ranging from direct methods [10]–[12], over alternating least squares methods [2], [13], [14] to all-at-once optimization methods [15]–[19]. Albeit fairly mature, rank estimation, ill-conditioned decompositions, and ill-posed optimization problems remain important issues [20]. On top of that, most algorithms quickly run into trouble for large-scale tensors, as both the memory and per-iteration computational complexity are linear in the number of entries in the tensor which increases exponentially in the order. These problems are a manifestation of the curse of dimensionality [21]–[24].

To overcome or at least reduce the difficulties related to large-scale tensors, many new strategies have emerged recently. A first category involves incomplete tensors where only a fraction of the elements of a tensor is known [15], [19], [21], [25]–[27]. The per-iteration complexity of these algorithms is linear in the number of known entries, which can be far lower than the number of entries in the full tensor. To take advantage of this, one can deliberately sample the tensor in only a few entries [21], [27]. A second category uses a similar idea for sparse tensors [28]–[30], where the per-iteration complexity is made linear in the number of nonzeros. Third, there are compression based algorithms. The tensor can, for example, be compressed multiple times using random projections [31] or using randomized SVDs [32]. Finally, some algorithms decompose subtensors and then recombine the factor matrices. In the grid PARAFAC method the tensor is subdivided in a grid and each block is decomposed separately. The CPD structured blocks are then used to efficiently compute the CPD of the original tensor [33]. Alternatively, blocks can be sampled and decomposed after which the factor matrices are merged [29].

Rapid developments in parallel and distributed computing have led to a number of new algorithms exploiting the architecture. GigaTensor [28] and PARACOMP [31] use the MapReduce paradigm. In grid PARAFAC all the blocks in

Nico Vervliet is supported by an Aspirant Grant from the Research Foundation – Flanders (FWO). This research is funded by (1) Research Council KU Leuven: C1 project c16/15/059-nD and CoE PFV/10/002 (OPTEC), (2) F.W.O.: project G.0830.14N and G.0881.14N, (3) the Belgian Federal Science Policy Office: IUAP P7/19 (DYSCO II, Dynamical systems, control and optimization, 2012-2017), (4) EU: The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC Advanced Grant: BIOTENSORS (no. 339804). This paper reflects only the authors' views and the Union is not liable for any use that may be made of the contained information.

Nico Vervliet is with the Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium, and with iMinds Medical IT, Leuven, Belgium. (e-mail: nico.vervliet@esat.kuleuven.be).

Lieven De Lathauwer is with both the Group of Science, Engineering and Technology, KU Leuven Kulak, E. Sabbelaan 53, B-8500 Kortrijk, Belgium and with the Department of Electrical Engineering (ESAT), KU Leuven, Kasteelpark Arenberg 10, B-3001 Leuven, Belgium, and with iMinds Medical IT, Leuven, Belgium. (e-mail: lieven.delathauwer@kuleuven-kulak.be).

the grid are decomposed simultaneously [33]. In [29] the parallelization potential is outlined for the ParCube method. An alternating direction method of multipliers (ADMoM) for a mesh type architecture is shown in [34].

The randomized block sampling CPD algorithm outlined in this paper mainly falls in the fourth, block decomposition, category. Each iteration a new random block of (not necessarily adjacent) entries is sampled, and the affected variables are updated using the previous results as initialization. Then this is repeated for a new block of entries. The main difference with other methods in this category, i.e., grid PARAFAC and ParCube, is the use of ideas from stochastic optimization rather than factorizing blocks (from a grid or randomly sampled) and then recombining the results in the end. After a short overview of stochastic optimization concepts in Section II, the actual algorithm is outlined in Section III. Section IV conceptually discusses the behavior of the algorithm. Two parameters are introduced: the step size restriction strategy and the block size. Their influence on the accuracy and decomposition time is illustrated and discussed in Section V.

Notation: Scalars, vectors, matrices and tensors are denoted by lower case (e.g. a), bold lower case (e.g. $\mathbf{a}$), bold upper case (e.g. $\mathbf{A}$) and calligraphic (e.g. $\mathcal{T}$) letters, respectively. Index sets will be denoted by the calligraphic letters $\mathcal{B}$ and $\mathcal{I}$ for block and tensor level index sets, respectively. The norm $||\cdot||$ is defined as the Frobenius norm. The Kronecker product, the Khatri–Rao product and the Hadamard product are denoted by $\otimes$, $\odot$ and $*$, respectively, and the mode-n tensor unfolding of $\mathcal{T}$ is given by $\mathbf{T}_{(n)}$ (see e.g. [1] for formal definitions). The conjugate of a complex value a is denoted by $\bar{a}$ and the transpose, conjugated transpose, inverse and pseudoinverse by $\cdot^{\mathrm{T}}$, $\cdot^{\mathrm{H}}$, $\cdot^{-1}$ and $\cdot^{\dagger}$, respectively.

II. Stochastic optimization

Randomization is often used to scale up algorithms for big data, e.g., in randomized linear algebra [35] and in stochastic optimization [36]. The methods discussed in this paper combine ideas from stochastic gradient descent (SGD) and block coordinate descent (BCD). In both cases the objective is to find the parameters $\mathbf{x}$ that minimize a function f:

$$\min_{\mathbf{x}} f(\mathbf{x}).$$

Before developing our randomized block sampling CPD method, some background on both methods is given.

SGD [37] is a simple yet powerful technique widely used in optimization and machine learning. It also appeared as the classical Least Mean Squares (LMS) filter which led to a large number of applications, e.g. in adaptive antenna arrays [38]. Suppose the objective function f can be decomposed in individual contributions f_n from each of the N_s data points:

$$f(\mathbf{x}) = \frac{1}{N_s}\sum_{n=1}^{N_s} f_n(\mathbf{x}).$$

An example of a decomposable function is the Frobenius norm error, e.g., to solve the overdetermined system $\mathbf{A}\mathbf{x} = \mathbf{b}$, the least squares formulation is

$$f(\mathbf{x}) = ||\mathbf{A}\mathbf{x} - \mathbf{b}||^2 = \sum_{n=1}^{N_s} (\mathbf{a}_n^{\mathrm{T}}\mathbf{x} - b_n)^2,$$

in which $\mathbf{a}_n^{\mathrm{T}}$ is the nth row of $\mathbf{A}$. In the SGD method, the gradient ∇f is estimated every iteration from a single random sample point n_k. The estimate of the parameters is then updated from the estimated gradient ∇f_{n_k} using

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f_{n_k}(\mathbf{x}_k),$$

in which α_k is the learning rate [36]. For matrix factorization, parallel implementations of SGD have been derived [39], [40]. Extensions to the CPD [41] and the (symmetrical) orthogonal decomposition in rank-1 terms [42] appeared recently.

In the case of the coordinate descent method a coordinate i_k is selected in every iteration. The parameter $\mathbf{x}$ is then updated using the exact gradient with respect to x_{i_k}

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \nabla_{i_k} f(\mathbf{x}_k)\mathbf{e}_{i_k},$$

in which α is the learning rate and $\mathbf{e}_{i_k}$ is the i_kth column of an identity matrix [36]. Random selection of the coordinates results in the optimal convergence rate in expectation [43]. The BCD algorithm selects a block of coordinates in every iteration.

Second-order information is often avoided because of the computational cost [36]. Despite this, the use of second-order information improves the convergence constants [44]. A couple of quasi-Newton approaches have emerged which approximate the Hessian by a diagonal matrix [45], [46] or using online LBFGS [47]. In the case of a CPD, the computation of a good Hessian approximation is relatively cheap, which makes a Gauss–Newton approach feasible.

III. CPD by randomized block sampling

The randomized block sampling (RBS) CPD algorithm is both a randomized BCD algorithm and a SGD method. In every iteration a block of variables is updated using an estimate of the gradient and the approximate Hessian based on a randomly sampled subblock of the tensor. The reason for this combination is the 'locality property' of the CPD: a single entry in an Nth order tensor of rank R affects only NR variables, while a block of size $B_1 \times \cdots \times B_N$ affects only $R\sum_{n=1}^{N} B_n$ variables. A CPD can be computed using a least squares approach which leads to the optimization problem

$$\min_{\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}} f \tag{2}$$

in which f is a decomposable function:

$$\begin{aligned} f &= \frac{1}{2}\left|\left|\mathcal{T} - \left[\!\left[\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(N)}\right]\!\right]\right|\right|^2 \\ &= \frac{1}{2}\sum_{i_1=1}^{I_1}\cdots\sum_{i_N=1}^{I_N}\left(t_{i_1\cdots i_N} - \sum_{r=1}^{R} a_{i_1 r}^{(1)} a_{i_2 r}^{(2)}\cdots a_{i_N r}^{(N)}\right)^2. \end{aligned}$$

Let a block be defined by the index sets $\mathcal{B}_1,\ldots,\mathcal{B}_N$, then the gradient ∇f is only nonzero for the variables $a_{i_n r}^{(n)}, i_n \in \mathcal{B}_n$, $n = 1,\ldots,N$.

Algorithm 1 gives a high level overview of the randomized block sampling CPD method. Every iteration k, a random block is sampled. Only the variables affected by this block are then updated. Contrary to the techniques outlined in Section II, a more informed update is computed using a relatively cheap approximation to the Hessian. The parameter Δ_k acts as the learning rate and is decreased in order to achieve convergence and to improve the accuracy as explained in Section IV and illustrated in the experiments. The sampling operator, the computation of the update and the learning rate selection are discussed in Subsections III-A, III-B and III-C, respectively. Subsection III-D introduces a new stopping criterion based on the Cramér–Rao bound.

Algorithm 1: Randomized block sampling CPD

while *not converged* **do**
  Randomly generate sample indices $\mathcal{B}_n \subseteq \mathcal{I}_n = \{1, \ldots, I_n\}$, $n = 1, \ldots, N$;
  Let $\mathcal{T}_{\text{sub}} = \mathcal{T}(\mathcal{B}_1, \ldots, \mathcal{B}_N)$, and $\mathbf{A}_{\text{sub}}^{(n)} = \mathbf{A}_k^{(n)}(\mathcal{B}_n, :)$, $n = 1, \ldots, N$;
  $\{\mathbf{A}_{\text{sub}}^{(n)}\} \leftarrow \text{update}(\mathcal{T}_{\text{sub}}, \{\mathbf{A}_{\text{sub}}^{(n)}\}, \Delta_k)$;
  Set $\mathbf{A}_{k+1}^{(n)} = \mathbf{A}_k^{(n)}$ and $\mathbf{A}_{k+1}^{(n)}(\mathcal{B}_n, :) = \mathbf{A}_{\text{sub}}^{(n)}$, $n = 1, \ldots, N$;
  $k \leftarrow k + 1$;
end

A. Sampling operator

Instead of randomly sampling blocks, we use a more involved sampling operator ensuring that every variable is updated at the same rate and allowing blocks to be decomposed in parallel. (A discussion of a parallel implementation is outside the scope of this paper.) The operator is illustrated in Figure 1.

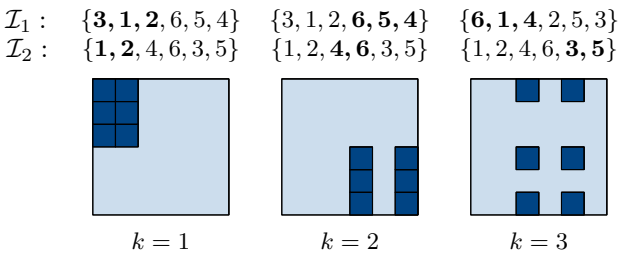

Fig. 1. Illustration of the block sampling operator for a second-order tensor of size 6×6 and block size 3×2. In iteration $k = 1$, the row index set $\mathcal{I}_1$ and the column index set $\mathcal{I}_2$ are shuffled, and first blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ are selected (bold). In iteration $k = 2$, the next block is selected. In iteration $k = 3$, the next column block $\mathcal{B}_2$ is sampled. The row index set $\mathcal{I}_1$ is permuted, as it has no blocks left, and $\mathcal{B}_1$ is the first block of the shuffled index set.

A sample block $\mathcal{T}_{\text{sub}}$ can be generated by selecting a random subset $\mathcal{B}_n$ of B_n indices from $\mathcal{I}_n = \{1, \ldots, I_n\}$. Each iteration N new subsets are generated. Let $Q_n = I_n/B_n$ be the number of blocks per dimension, then after every Q_n iterations, the elements in the index set $\mathcal{I}_n$ are permuted. The indices $\mathcal{B}_n$ are now selected consecutively from $\mathcal{I}_n$, restarting at the first block when $\mathcal{I}_n$ is permuted. More formally, the index sets for $k = 1$ are $\mathcal{B}_n = \mathcal{I}_n(1 : B_n)$, for $k = 2$, $\mathcal{B}_n = \mathcal{I}_n(B_n + 1 : 2B_n)$, and for general $k \leq Q_n$, $\mathcal{B}_n = \mathcal{I}_n((k-1)B_n + 1 : kB_n)$. If $k = Q_n + 1$ no blocks are left and $\mathcal{I}_n$ is shuffled, after which the first indices again define the new index block, i.e. $\mathcal{B}_n = \mathcal{I}_n(1 : B_n)$. In general $\mathcal{B}_n = \mathcal{I}_n((l_n - 1)B_n + 1 : l_n B_n)$, $l_n = \text{mod}(k - 1, Q_n) + 1$ and $\mathcal{I}_n$ is permuted every time that $\text{mod}(k - 1, Q_n) = 0$. When Q_n is not an integer, there are two choices: either the last variables are ignored and $\mathcal{I}_n$ is shuffled when no full block is available, or a smaller block sample is determined by the remaining indices.

The block size and the randomization play important roles in the algorithm as they influence the robustness, the total computation time and the attainable accuracy. Section IV discusses this influence and the effect will be thoroughly tested in the experiments in Section V.

B. Computing the update

Any CPD algorithm can be used to compute the update in Algorithm 1. Two variants are developed here: an alternating least squares (ALS) version and a nonlinear least squares (NLS) version.

ALS with step size restriction: ALS is a well-known optimization technique to solve the least squares problem (2) (see e.g. [2], [13], [14]). By fixing all but one factor matrix (say $\mathbf{A}^{(n)}$), Equation (2) becomes a linear least squares problem in the factor matrix $\mathbf{A}^{(n)}$:

$$\min_{\mathbf{A}^{(n)}} \frac{1}{2} \left|\left| \mathbf{T}_{(n)} - \mathbf{A}^{(n)} \mathbf{V}_k^{(n)^{\mathrm{T}}} \right|\right|^2, \tag{3}$$

in which $\mathbf{V}_k^{(n)} = \mathbf{A}_k^{(N)} \odot \cdots \mathbf{A}_k^{(n+1)} \odot \mathbf{A}_{k+1}^{(n-1)} \odot \cdots \odot \mathbf{A}_{k+1}^{(1)}$. The least squares problem (3) has an exact solution, but in order to introduce the step size parameter α, we compute the update explicitly as $\mathbf{A}_{k+1}^{(n)} = \mathbf{A}_k^{(n)} + \alpha \mathbf{P}_k$. The gradient and Hessian of the cost function in (3), evaluated in iteration k, are given by $\mathbf{G}_k = \mathbf{A}_k^{(n)} \overline{\mathbf{W}}_k^{(n)} - \mathbf{T}_{(n)} \overline{\mathbf{V}}_k^{(n)}$ and $\mathbf{H}_k = \mathbf{W}_k^{(n)} \otimes \mathbf{I}_{I_n}$, respectively, in which $\mathbf{W}_k^{(n)} = (\mathbf{V}_k^{(n)})^{\mathrm{H}} \mathbf{V}_k^{(n)}$ is computed efficiently as $\mathbf{W}_k^{(n)} = \mathbf{A}_{k+1}^{(1)^{\mathrm{H}}} \mathbf{A}_{k+1}^{(1)} * \cdots * \mathbf{A}_{k+1}^{(n-1)^{\mathrm{H}}} \mathbf{A}_{k+1}^{(n-1)} * \mathbf{A}_k^{(n+1)^{\mathrm{H}}} \mathbf{A}_k^{(n+1)} * \cdots * \mathbf{A}_k^{(N)^{\mathrm{H}}} \mathbf{A}_k^{(N)}$ and $\mathbf{I}_{I_n}$ is the $I_n \times I_n$ identity matrix. The optimal step $\text{vec}(\mathbf{P}_k)$ is given by $\mathbf{H}_k \text{vec}(\mathbf{P}_k) = -\text{vec}(\mathbf{G}_k)$ which is equivalent to $\mathbf{P}_k \mathbf{W}_k^{(n)^{\mathrm{T}}} = \mathbf{P}_k \overline{\mathbf{W}}_k^{(n)} = -\mathbf{G}_k$ using properties of the Kronecker product. Therefore, we find

$$\mathbf{P}_k = \mathbf{T}_{(n)} \overline{\mathbf{V}}_k^{(n)} \left(\overline{\mathbf{W}}_k^{(n)} \right)^{-1} - \mathbf{A}_k.$$

Finally, the updated factor matrix $\mathbf{A}_{k+1}^{(n)}$ is given as

$$\mathbf{A}_{k+1}^{(n)} = (1 - \alpha) \mathbf{A}_k^{(n)} + \alpha \mathbf{T}_{(n)} \overline{\mathbf{V}}_k^{(n)} \left(\overline{\mathbf{W}}_k^{(n)} \right)^{-1}.$$

When $\alpha = 1$, the commonly used ALS update is obtained. Here we use $\alpha = \Delta_k$ to control the step lengths.

NLS with step size restriction: An optimum of (2) can also be found using all-at-once-optimization, and in the case of NLS problems, the Gauss–Newton algorithm can be used. Let $\mathbf{x}$ be the concatenation of the vectorized factor matrices $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. The Gauss–Newton algorithm solves (2) by linearizing $\mathcal{F}(\mathbf{x}) = \mathcal{T} - [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}]\!]$ in every iteration and solving

$$\min_{\mathbf{p}_k} \frac{1}{2} \left|\left|\text{vec}\left(\mathcal{F}(\mathbf{x}_k)\right) - \mathbf{J}_k \mathbf{p}_k\right|\right|^2$$

in which the step $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{J}_k$ is the Jacobian matrix $\partial \text{vec}\left(\mathcal{F}\right) / \partial \mathbf{x}^{\text{T}}$ evaluated at $\mathbf{x}_k$. The step size can be restricted by adding a constraint on the step size

$$\min_{\mathbf{p}_k} \frac{1}{2} \left|\left|\text{vec}\left(\mathcal{F}(\mathbf{x}_k)\right) - \mathbf{J}_k \mathbf{p}_k\right|\right|^2 \text{ s.t. } ||\mathbf{p}_k|| \leq \Delta_k. \quad (4)$$

This is a similar formulation as in trust region algorithms [48]. The main difference is that Δ_k is not updated based on the trustworthiness of the linearized model, but explicitly set by the stochastic algorithm. Similar strategies as for trust regions can be used to solve (4) approximately. In our implementation we use the dogleg method, which approximately finds a solution to (4) using the Gauss–Newton direction and the steepest descent direction [16], [48]. Usually we assume the step constraint is relative, i.e. $||\mathbf{p}_k|| \leq \Delta_k ||\mathbf{x}_0||$, where $\mathbf{x}_0$ is the initial guess. Note that, contrary to the ALS variant, Δ_k is only an upper bound of the step size.

C. Step size selection

In stochastic optimization the selection of the step size is an important element in the convergence of the algorithm. If the step size decreases too slowly, a lot of computation time is wasted, whereas one may not reach the optimum in time if the step size decreases too fast. In a signal processing context, the data is often perturbed by noise, rendering step size selection schemes based on the function value hard. Often a fairly good solution can be attained without restricting the step size [49], which is also what we experience in the experiments. Therefore a search-then-converge strategy can be used [50] in which the step size is large and constant for a number of iterations and then gradually decreased. For simplicity we use a two step strategy:

$$\Delta_k = \begin{cases} \delta_0 & \text{if } k < K_{\text{search}} \\ \delta_{K_{\text{search}}} \cdot \alpha^{(k - K_{\text{search}})/Q} & \text{if } k \geq K_{\text{search}} \end{cases},$$

in which $Q = \max_n I_n / B_n$. The initial δ_0 can be quite large, e.g., 0.8 for the NLS method, and should be 1 for the ALS method. The parameter $\delta_{K_{\text{search}}}$ is generally smaller than δ_0 to speed up convergence. In the case of NLS, the restriction Δ_k is used on the relative step size $||\mathbf{p}_k|| \,/\, ||\mathbf{x}_0||$. The shrinkage factor $\alpha < 1$ and K_{search} have to be determined experimentally. As explained in Section IV, K_{search} should be set large enough such that the algorithm has converged to the neighborhood of the optimum after K_{search} iterations, and α should be large, e.g. 0.99, to improve the accuracy. Tuning the parameters can shorten the computation time considerably. The parameters can often be found by decomposing a smaller, representative subtensor first [51]. This is illustrated in Section V-B.

D. Stopping criterion

We propose a new stopping criterion based on the estimated Cramér–Rao bound. Conventional stopping criteria in CPD algorithms involve either an evaluation of the objective function or depend on the (relative) step size. Both criteria can be used here as well, but some remarks have to be made. The function value is known to decrease only down to the noise level, which causes premature convergence in low SNR cases. Continuing to iterate often improves the solution (as shown in the convergence plots in Sections V-A and V-B). Moreover, we can only evaluate the function in subblocks as computing the function value for the full tensor is unfeasible for large-scale tensors. The function value, which is computed for a block, does not decrease monotonically, as one block may have a better fit than another one. It is also rather difficult to use the step size, as the step length is explicitly restricted by Δ_k.

The Cramér–Rao bound on the other hand takes the noise estimate into account as well as step size. The Cramér–Rao bound $\mathbf{C}$ gives a lower bound on the covariance matrix of the estimators $\hat{\mathbf{A}}^{(n)}$ of the variables $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$. In the stopping criterion, we only consider the estimated lower bounds on the variances which can be found on the diagonal of $\mathbf{C}$. (In other words, we do not use the covariances.) Let $\mathbf{c} = \text{diag}(\mathbf{C})$ be the vector of length $R \sum_{n=1}^{N} I_n$ that contains these variances, then we define $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$, as the $I_n \times R$ matrix with the lower bounds on the variances, i.e. $\mathbf{C}^{(n)} = \text{reshape}(\mathbf{c}(J_{n-1} + 1 : J_n), I_n, R)$, $J_n = R \sum_{k=1}^{n} I_n$. The Cramér–Rao bound then implies that for a noisy observation of a tensor generated by $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$, the estimator lies with a probability of 99.7% in the interval $\mathbf{A}^{(n)} \pm 3\mathbf{\Sigma}^{(n)}$, where $\mathbf{\Sigma}^{(n)} \geq \sqrt{\mathbf{C}^{(n)}}$ in which both $\geq$ and the square root are defined element-wise. If the current estimate $\mathbf{A}_k^{(n)}$, $n = 1, \ldots, N$ is close to a stationary point, we argue that it makes little sense to set steps that are small compared to the lower bound on the variance as the noise makes the result unreliable. Let us define the mean absolute difference between the estimated variables in iteration k and $k - K_{\text{CRB}}$ relative to the Cramér–Rao bound as

$$D_{\text{CRB}} = \frac{1}{R \sum_n I_n} \sum_{n=1}^{N} \sum_{i=1}^{I_n} \sum_{r=1}^{R} \frac{\left|\mathbf{A}_k^{(n)}(i,r) - \mathbf{A}_{k-K_{\text{CRB}}}^{(n)}(i,r)\right|}{\sqrt{\mathbf{C}^{(n)}(i,r)}}.$$

The algorithm is said to have converged if

$$D_{\text{CRB}} < \gamma \quad (5)$$

and γ is a constant, e.g., 0.5. The window K_{CRB} over which the change is computed, is introduced for robustness. Namely, if the step size is very small, but the steps proceed in the same direction, the overall difference over multiple steps can be large. On the other hand, if the algorithm repeatedly jumps over the optimum, the net difference will be small. Note that the scaling indeterminacies are taken care of: when the norm of a factor vector $\mathbf{a}_r^{(n)}$ increases, the corresponding entries in $\mathbf{C}^{(n)}$ increase appropriately.

The Cramér–Rao bound for a CPD is given by

$$\mathbf{C} = \sigma^2 (\mathbf{J}^{\text{H}} \mathbf{J})^{-1},$$

where $\mathbf{J}^{\text{H}}\mathbf{J}$ is the Gramian of the Jacobian of (2) [25], [52]. The Gramian is not invertible, however, as it has at least $(N-1)R$ singular values equal to zero due to scaling indeterminacies. These indeterminacies can be resolved by fixing $(N-1)R$ entries [52] [25], or the pseudoinverse inverse can be used instead, as is illustrated here. The observation that $\mathbf{J}^{\text{H}}\mathbf{J}$ can be factored as $\mathbf{G} + \mathbf{Z}\mathbf{K}\mathbf{Z}^{\text{H}}$ with $\mathbf{G}$ and $\mathbf{Z}$ block diagonal [17], has led to an efficient inversion algorithm [25]. Using a similar derivation as in [25], the Cramér–Rao bound can be computed using the pseudoinverse:

$$\begin{aligned} \mathbf{B} &= \mathbf{K}\left(\mathbf{I}_{NR^2} + \mathbf{Z}^{\text{H}}\mathbf{G}^{-1}\mathbf{Z}\mathbf{K}\right)^{\dagger} \\ \frac{1}{\sigma^2}\mathbf{C} &= \mathbf{G}^{-1} - \mathbf{G}^{-1}\mathbf{Z}\mathbf{B}\mathbf{Z}^{\text{H}}\mathbf{G}^{-1}. \end{aligned} \tag{6}$$

By exploiting that $\mathbf{G}$ and $\mathbf{Z}$ are block diagonal and that only the estimated lower bounds on the variances $\mathbf{C}^{(n)}$ are used, this bound can be computed very efficiently as long as R is low. (The computation cost is governed by the pseudoinverse of the $NR^2 \times NR^2$ matrix in Equation (6).) If the computational cost of evaluating $\mathbf{C}^{(n)}$, $n = 1, \ldots, N$ becomes large compared to the amount of useful work, a computationally (much) cheaper variant can be used. As shown in [53], the inverse of the diagonal of the covariance matrix is larger than the diagonal elements of the inverse of the covariance matrix, i.e., let $\mathbf{F} = \mathbf{C}^{-1}$ then $\mathbf{C}(i,i) \geq (\mathbf{F}(i,i))^{-1}$, $i = 1, \ldots, R\sum_{n=1}^{N} I_n$. This means that only the diagonal of $\mathbf{J}^{\text{H}}\mathbf{J}$ needs to be inverted, which contains only NR unique entries [16], [25] . This new bound is a, possible loose, lower bound for the Cramér–Rao bound [53], leading to a more severe stopping criterion. This can be compensated for by adjusting γ. In our experience, the entry-wise bound is usually a factor 1 to 5 smaller than the exact bound.

In practice the true underlying variables are unknown, as is the noise level σ. Instead of the true variables, the estimated variables are used as they are expected to converge to the real variables. The noise variance can be estimated from the previous K_{noise} function values f_k using

$$\sigma^2 = \frac{2}{K_{\text{noise}}(\prod_n B_n) - 1} \sum_{l=k-K_{\text{noise}}+1}^{k} f_l. \tag{7}$$

Recall that $f_k = 0.5 \left|\left| \mathcal{T}_{\text{sub}} - \left[\!\left[\mathbf{A}_{\text{sub}}^{(1)}, \ldots, \mathbf{A}_{\text{sub}}^{(N)} \right]\!\right] \right|\right|^2$ for the sample block used in iteration k. By dividing $2f_k$ by the number of elements $\prod_n B_n$ in a sample block, we obtain the average squared error which indeed defines the estimator of the variance. The minus one in (7) ensures that the estimator is unbiased. The noise window K_{noise} should be at least $\max_n Q_n$ such that all variables are accounted for in the noise computation. A large K_{noise} provides a better estimate of σ^2. Initially, when the solution is far off the optimum, the corresponding f_k do not provide a good estimate for the noise, hence K_{noise} should not be too large in order to ignore these initial values.

The Cramér–Rao bound used here is derived under the assumption of exact rank and Gaussian i.i.d. noise on the tensor. If these conditions do not hold, the Cramér–Rao bound may no longer be a lower bound for the variances. The estimated bound can suggest a good stopping criterion nonetheless.

IV. Conceptual discussion

We now discuss the behavior of the algorithm as a whole. The goal of this section is to give some intuition on why the algorithm works rather than a rigorous proof. To keep ideas simple, we restrict our discussion to well behaved problems involving tensors with an exact CPD structure of a known rank R which is low compared to the dimensions, and possibly perturbed by zero-mean i.i.d. noise such that the SNR is high enough. For these tensors, blocks with most dimensions larger than the rank can be selected. Note that we do not make claims for difficult decompositions, e.g., tensors that only just satisfy uniqueness conditions, tensors with very high rank or tensors with a low SNR. For these tensors identifiability can potentially be lost, or the estimation accuracy can be decreased when randomized block sampling is used.

The optimal estimator for a particular noise realization of a rank-R tensor $\mathcal{T} = \left[\!\left[\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)} \right]\!\right]$ is denoted by $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$. Note that the minimizers $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$ of optimization problem (2) are perturbed w.r.t. $\mathbf{A}^{(n)}$, $n = 1, \ldots, N$ because of the perturbation of the tensor by that particular noise term. The progress of the algorithm can be split in two phases: the search phase during which the step size is unrestricted, and the converge phase with step restriction. During the unrestricted phase, the algorithm converges to a neighborhood of a (local) optimum in which the iterates eventually jump around. The restricted phase can be seen as a variance reduction strategy which improves the accuracy of the solution and leads to an estimate of $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$. Proving convergence to an optimum is still an open problem for general rank-R tensors. For truly big tensors, we often have to make compromises, however, and we therefore aim at finding a reasonably good solution. The experiments indicate that the RBS CPD algorithm succeeds in finding such a good approximation. For the remainder of this section, we focus on the NLS variant. The reasoning can be extended to the ALS variant.

A. Unrestricted phase

During this first phase, the algorithm tries to improve the initial guess $\mathbf{x}_0$ by moving the current iterate to a neighborhood of an optimum. We will now show this by leveraging uniqueness of blocks to uniqueness of the full tensor. We explain that near the end of this phase, the iterates $\mathbf{x}_k$ are jumping around in the neighborhood of an optimum because of the block sampling and the convergence properties of NLS type algorithms close to optima. The parameter K_{search} is chosen to mark the end of this phase.

We only consider blocks with most dimensions larger than the rank of the full tensor. First, consider the noiseless case. In this case, the CPD of a block with dimensions larger than the rank is likely to be unique; the solution can actually be computed using a generalized eigenvalue decomposition [54]. The CPD of the full tensor inherits its uniqueness from the uniqueness of the sub-CPDs. If we add noise to the tensor

such that the SNR remains high enough, we generally do not expect major problems to find the optimum for a block. We expect that the optimal CPD for the full tensor can now again be found from the sub-CPDs.

Using the reasoning above, a step computed from a block is generally expected to be well-conditioned in the sense that the optimum of the sub-CPDs will lead to the optimum for the full tensor. However, particular blocks can result in ill-conditioned steps. On the other hand, as only one step is computed for each block, the effect of this ill-conditioned block on the optimization process is limited. Due to the randomization, the probability of sampling many ill-conditioned blocks one after another is small as the total number of blocks is $\prod_{n=1}^{N} \binom{I_n}{B_n}$ and ill-conditioned blocks are often local phenomena, e.g., a part of a tensor that has lower rank (see Section V-C for an example).

The NLS variant of the RBS CPD algorithm is a second-order algorithm. Each iteration, $\mathbf{x}_k$ is updated as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma_k \mathbf{B}_k \mathbf{g}_k,$$

in which $\mathbf{B}_k = (\mathbf{J}_k^{\mathrm{H}} \mathbf{J}_k)^{\dagger}$ is a computationally cheap positive semidefinite approximation of the inverse Hessian. Because the restriction is not effective in this phase, the step size γ_k is chosen such that the quadratic model is minimized in each step, i.e.,

$$\gamma_k = \min \left(\frac{\mathbf{g}_k^{\mathrm{H}} \mathbf{g}_k}{\mathbf{g}_k^{\mathrm{H}} (\mathbf{J}_k^{\mathrm{H}} \mathbf{J}_k) \mathbf{g}_k}, 1 \right).$$

A property of second-order algorithms is their fast convergence near optima. At the end of the unrestricted phase, the algorithm is assumed to have converged to a neighborhood of an optimum. Therefore, the NLS algorithm (approximately) finds the optimum for the next block $k+1$ in one step regardless of the previous block. By consequence, $\mathbf{x}_{k+1}$ depends only on block $k+1$ and not on $\mathbf{x}_k$. This causes the algorithm to jump around in this neighborhood as each block has it own optimum because of the noise. The uncertainty on the estimates (which is determined by their variances) is larger for a block than the uncertainty that can be expected for the full tensor, as the dimensions of a block are (far) smaller than the dimensions of the full tensor. (This can be verified by comparing the Cramér–Rao bound for both the block and the full tensor, as this gives a lower bound on their variances.) This limits the accuracy of the solution in the unrestricted phase. In the following section, we show that step restriction reduces this loss in accuracy. Note that the next phase is only relevant for noisy tensors, as the uncertainty for exact tensors is zero.

B. Restricted phase

At the end of the first phase, the iterates $\mathbf{x}_k$ are jumping around in a neighborhood of an optimum. By applying a step length restriction, we now show that the uncertainty can be reduced. Next, we illustrate how a good choice for the step restriction schedule reduces the variance quickly. Finally, we show how the Cramér–Rao bound stopping criterion interacts with the variance reduction.

We focus, without loss of generality, on one variable, say $a_k = \mathbf{A}_k^{(1)}(1,1)$. The discussion can be generalized to updates of all variables $\mathbf{x}_k$, but for the clarity of the presentation we only present the results for a single variable. We assume the permutation and scaling indeterminacies are resolved. (We also ignore the fact that the variable is only updated every Q_1 iterations.) Without step restriction, the variance of a_k is given by σ_b^2. As explained in the previous section, the NLS algorithm approximately finds the optimal value $\tilde{a}_{k+1}$ for the unrestricted case for block $k+1$ in one step, i.e., $\tilde{a}_{k+1} = a_k + p_k$, where p_k denotes the unrestricted step. As $\tilde{a}_{k+1}$ is the optimal value for the next unrestricted block problem, its variance is again σ_b^2. Now, we impose the step restriction γ_k

$$a_{k+1} = a_k + \gamma_k p_k = (1-\gamma_k) a_k + \gamma_k \tilde{a}_{k+1}.$$

The estimate a_{k+1} can be seen as a running average over all $k+1$ estimates $\tilde{a}_l$, $l = 1, \ldots, k+1$. Blocks k and $k+1$ are selected independently, and are both perturbed by mutually independent noise terms. As $\tilde{a}_{k+1}$ is the optimum for block $k+1$, $\tilde{a}_{k+1}$ is independent from a_k. Therefore, the variance of a_{k+1} is

$$\begin{aligned} \operatorname{Var}(a_{k+1}) &= (1-\gamma_k)^2 \operatorname{Var}(a_k) + \gamma_k^2 \operatorname{Var}(\tilde{a}_{k+1}) \\ &= (1-\gamma_k)^2 \operatorname{Var}(a_k) + \gamma_k^2 \sigma_b^2, \end{aligned}$$

Lemma 1. *Setting $\gamma_k \in (0,1)$ reduces the variance Var(a_{k+1}). For constant $\gamma_k = \gamma$, the variance asymptotically goes to zero as $\gamma \to 0$.*

Proof. In this proof, we keep $\gamma_k = \gamma$ constant. Define $\beta = (1-\gamma)^2$. Let a_0 be the guess at the end of the unrestricted phase, and $k=1$ the first iteration in the restricted phase, then

$$\begin{aligned} \operatorname{Var}(a_1) &= \beta \sigma_b^2 + \gamma^2 \sigma_b^2 && k = 1 \\ \sigma_b^{-2} \operatorname{Var}(a_k) &= \beta^k + \gamma^2 \sum_{l=0}^{k-1} \beta^l && k \geq 1 \\ \sigma_b^{-2} \operatorname{Var}(a_k) &= \frac{1}{1-\beta} \gamma^2 = \frac{\gamma}{2-\gamma} && k \to \infty. \end{aligned}$$

For $\gamma \in (0,1)$, we have $\gamma/(2-\gamma) < 1$, thus the variance is reduced. The derivative w.r.t. γ is $2/(2-\gamma)^2 > 0$ for $\gamma \in (0,1)$, hence decreasing γ reduces the asymptotic variance, and, for $\gamma \to 0$, Var$(a_k) \to 0$. □

The proof above assumes constant γ_k. For decreasing step sizes γ_k, the expressions get more involved. This enables us to reduce the variance faster, as explained in the next paragraph.

Step restriction schedules: We now move on to decreasing step size γ_k. Figure 2 shows the evolution of the variance for different step restriction schedules. In Section III-C we defined exponentially decreasing step schedules of the form $\delta_{K_{\text{search}}} \alpha^k$. Setting α closer to 1 reduces the variance at a slower rate, but has a larger variance reduction effect.

Figure 2 also shows the more conventional $1/k$ restriction scheme which reduces the variance to zero and may therefore seem more interesting. A key factor in the NLS algorithm is the iteration where the restriction becomes active. (Remember that Δ_k is an upper bound on the step size.) Suppose this occurs K_{active} iterations after the start of the restricted phase, then the $1/k$-type restriction behaves as $K_{\text{active}}/(k + K_{\text{active}})$, which reduces the variance far slower. The proposed exponentially decreasing step sizes are less sensitive to the choice of

K_{search} (and hence K_{active}) as the variance is reduced at a more constant rate.

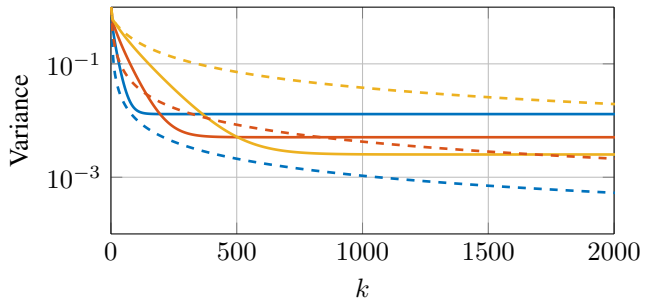

Fig. 2. Variance after k reduction steps for exponentially decreasing steps 0.95^k (——), 0.98^k (——) and 0.99^k (——), and inverse steps $1/k$ (- - -), $10/(k+10)$ (- - -), and $100/(k+100)$ (- - -).

Stopping criterion: We have showed that the variance of the estimate $\text{Var}(a_{k+1})$ can asymptotically be reduced to zero for certain choices of step restrictions γ_k. As indicated in the beginning of this section, the estimates $\hat{\mathbf{A}}^{(n)}$, $n = 1, \ldots, N$ are not identical to the true $\mathbf{A}^{(n)}$ because of perturbations caused by noise. After a number of iterations, the step restriction can reduce the 'algorithmic' uncertainty of a_{k+1} below the 'intrinsic' uncertainty due to noise on the tensor. The Cramér–Rao bound stopping criterion takes this into account.

V. Analysis and Experiments

The step size restriction strategy and the selection of the block size are important parameters in the algorithm. The following experiments illustrate their effect on the number of data accesses, computation time and accuracy. Except for the last experiment where hazardous gasses are analyzed, we focus on synthetically generated data. All tensors are generated using random factor matrices in which the elements are drawn from either the standard normal distribution $\mathcal{N}(0,1)$ or the uniform distribution $\mathcal{U}(0,1)$. The former distribution results in well-conditioned factor matrices, as the expected angle between the factor vectors is $90°$. When using the latter distribution, the expected angle is $42°$ which results in less well-conditioned factor matrices. If noisy tensors are used, the elements are perturbed by i.i.d. Gaussian noise which is scaled to obtain a given signal-to-noise ratio (SNR). To initialize the algorithm, random factor matrices are generated from the same distribution as the original factor matrices, and the exact rank is used. Finding the rank of a tensor can be a hard problem in practice and the rank is often determined by prior knowledge or multiple attempts with different ranks. These techniques can be used here as well, and will not be discussed further. In all experiments, two random initializations are used for each Monte Carlo experiment and the best result is retained. The algorithms are implemented in Matlab and make use of (adapted versions of) existing Tensorlab routines for the update step, more in particular `cpd_als` and `cpd_nls` with the Gauss–Newton solver with dogleg trust regions [16], [55]. Unless explicitly stated otherwise, the algorithm is said to have converged if the relative step size is smaller than 10^{-15} or the Cramér–Rao bound criterion (5) is met for $\gamma = 0.1$ and $K_{\text{CRB}} = 2$. To measure the accuracy of the estimates $\hat{\mathbf{A}}^{(n)}$ the CPD error E_{CPD} compared to the original factor matrices $\mathbf{A}^{(n)}$ is used, where

$$E_{\text{CPD}} = \max_n \frac{\left|\left|\hat{\mathbf{A}}^{(n)} - \mathbf{A}^{(n)}\right|\right|}{\left|\left|\mathbf{A}^{(n)}\right|\right|},$$

in which the scaling and permutation ambiguities have been resolved (using `cpderr` [55]). The timing experiments are performed on a standard laptop (quad core i7-4800MQ @ 2.70 GHz, 16 GB RAM, 256 GB Toshiba THNSNH25 SSD) running Matlab 2014b on Ubuntu 14.04.

A. Influence of the step size adaptation

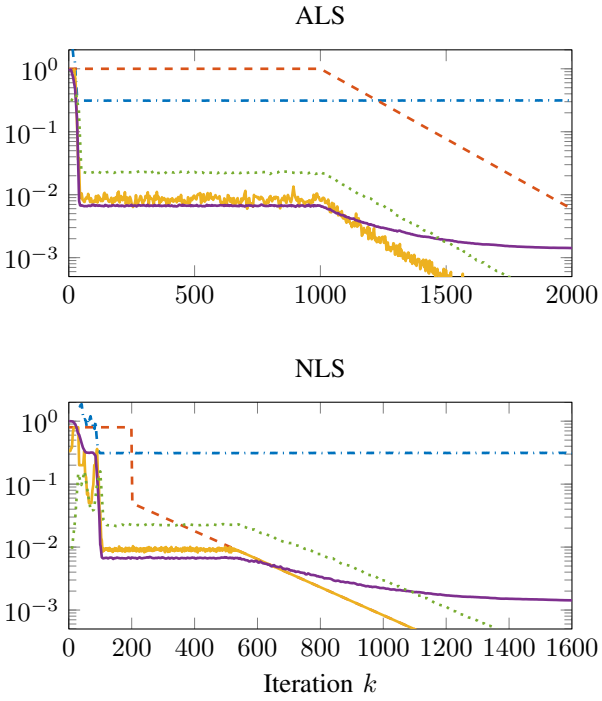

Fig. 3. Typical convergence plot for ALS and NLS. As soon as the (relative) step size (——) is restricted by Δ (- - -), the CPD error E_{CPD} (——) decrease further. In case of ALS $\alpha = \Delta_k$ influences the step size, but does not impose a direct constraint which explains why the curves do not touch. The mean error (-·-·-) and the change relative to the Cramér–Rao bound scaled by a factor 10^{-3} (·····) are also shown.

First we analyze the effects of step size restriction. As an example a rank $R = 10$ tensor of size $250 \times 250 \times 250$ is generated using well-conditioned factor matrices with entries drawn from the standard normal distribution $\mathcal{N}(0,1)$. The SNR is set to 20 dB. Blocks of size $50 \times 50 \times 50$ are sampled. The step restriction schedule first keeps Δ_k constant for $K_{\text{search}} = 200$ iterations in the NLS case and $K_{\text{search}} = 1000$ iterations in the ALS case. After the search phase, the step size is decreased exponentially using $0.95^{(k-K_{\text{search}})/10}$. (For the NLS variant, there is also a jump from 0.8 to 0.05 such that the restriction becomes effective sooner.) The convergence plots for both the NLS and the ALS variants are shown in Figure 3. When $k < K_{\text{search}}$ both variants converge to a

reasonably accurate solution, but the relative step size and the change in variables compared to the Cramér–Rao bound remain relatively high. This indicates that the algorithm is jumping around the optimum. In the NLS case, there is no effective restriction as the relative step size is always smaller than $\Delta_k = 0.8$. In the ALS case $\Delta_k = 1$, which means common unrestricted ALS iterations are performed. As the exact solution is known, we can monitor the CPD error E_{CPD}, and it has a similar behavior as the step size. When Δ_k starts restricting the step size (around $k = 550$ for NLS and $k = 1000$ for ALS), the step size, the change relative to the Cramér–Rao bound and E_{CPD} start decreasing again. Note that the function value (mean error in Figure 3) appears to be constant after few iterations at a level determined by the noise. However, a lot of progress can still be made in terms of accuracy as shown by the E_{CPD} curve. Figure 4 shows the effect of restricting the step size more clearly for uniformly distributed factor matrices and the NLS variant. By carefully choosing the step restriction schedule a solution as accurate as the full tensor solution can be attained in little time if the SNR is high enough. This is illustrated further in Section V-B.

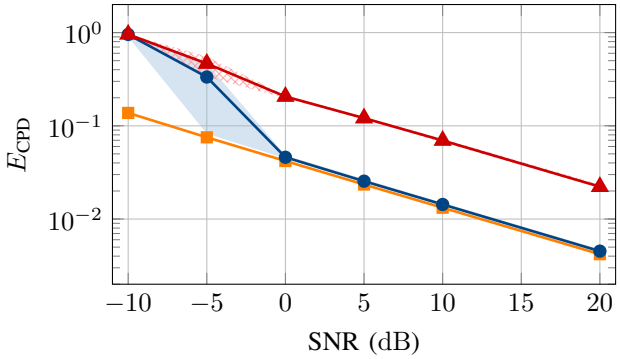

Fig. 4. Median accuracy over 50 Monte Carlo experiments and two initializations without (—▲—) and with (—●—) step restriction for different noise levels using the NLS variant and uniformly distributed factor matrices. As a baseline the accuracy using the full tensor is given (—■—). The shadings indicate the minimum and maximum accuracy achieved over all experiments.

Typically, ALS performs well on well-conditioned data like the normally distributed factor matrices used in the previous experiment. This is illustrated in Figure 5. Here both normally distributed and uniformly distributed factor matrices are used. (All other parameters are the same as in the previous experiment.) For normally distributed factor matrices, ALS achieves the same median accuracy as NLS, but sometimes it does not converge within the maximum of 2000 iterations, hence the large spread. NLS on the other hand always converges. When using uniformly distributed random factor matrices, the problem is more difficult. ALS now only converges to a good solution if the SNR is high, and even then ALS often fails to find a good solution. The NLS variant always finds a good solution if the SNR is not too low. Table I shows the results in more detail for a SNR of 10 dB. If ALS converges, a good solution is found quickly. For uniformly distributed factor matrices, the difference in computation time between ALS and NLS becomes small. ALS consumes much more data, however: while NLS often converges before all entries are accessed, ALS processes every element multiple times. The choice between the ALS and NLS version hence depends on the condition of the data and the cost of accessing or generating data.

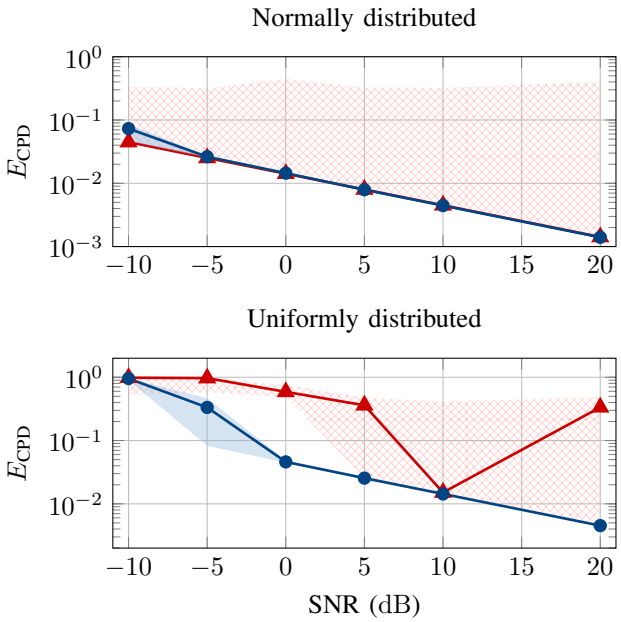

Fig. 5. Median accuracy over 50 Monte Carlo experiments and two initializations for ALS (—▲—) and NLS (—●—) for different noise levels and type of factor matrices: normally distributed (top) and uniformly distributed (bottom). The shadings indicate the minimum and maximum accuracy achieved over all experiments.

TABLE I
FOR NORMALLY AND UNIFORMLY DISTRIBUTED FACTOR MATRICES, THE PERCENTAGE OF EXPERIMENTS IN WHICH THE ALGORITHM CONVERGED, THE MEDIAN PERCENTAGE OF DATA ENTRIES SAMPLED AND THE MEDIAN RUN TIME ARE GIVEN FOR THE SNR = 10 dB CASE. WHEN ALS CONVERGES, IT IS USUALLY FAST, BUT NEEDS A LOT OF SAMPLES. NLS ALWAYS CONVERGES AND OFTEN DOES NOT ACCESS THE FULL TENSOR.

Distribution	Method	Conv. (%)	Data (%)	Time (s)
Normal	ALS	86	748.0	14.45
	NLS	100	56.6	30.26
Uniform	ALS	76	253.0	23.10
	NLS	100	42.4	26.57

B. *Step size selection for an* 8 TB *tensor*

The previous analysis showed the importance of the step selection strategy. Now we illustrate how to choose this strategy for a $1000 \times 1000 \times 1000 \times 1000$ tensor, which would consume 8 TB of memory if generated. The rank $R = 20$ tensor is generated from uniformly distributed random factor matrices. In the first factor matrix, we set the top half of the first 19 columns to zero, which means the top half of the tensor has only rank 1. The tensor is then perturbed by Gaussian i.i.d. noise such that the SNR is 20 dB. Blocks of size $40 \times 40 \times 40 \times 40$ are used and are generated when needed. To determine the step size, a small random subtensor of size $80 \times 80 \times 80 \times 80$ is sampled from the tensor, hence $Q = 2$.

The following step size strategy is used to start:

$$\Delta_k = \begin{cases} 0.8 & \text{if } k < 100 \\ 0.1 \cdot (0.85)^{(k-100)/2} & \text{if } k \geq 100 \end{cases}.$$

Figure 6 shows the resulting convergence behavior. The step size restriction becomes effective only after 140 iterations, while the convergence stagnates after 25 iterations, which is confirmed by the CPD error. The computation time can be reduced by restricting the step size earlier, e.g., at $k = 30$. Additionally, we will reset Δ_k to 0.01 at $k = 30$ because the step size drops rather slowly, resulting in only small decreases in the change relative to the Cramér–Rao bound.

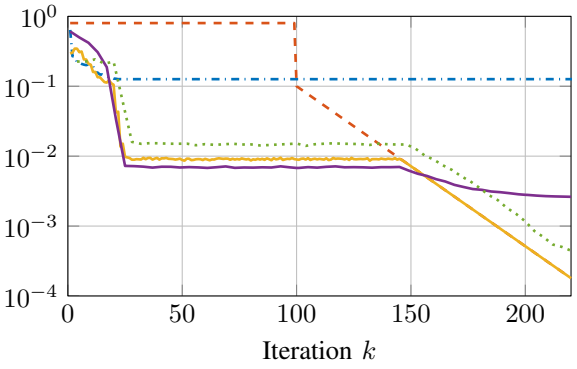

Fig. 6. Decomposition of a random sample of size $80 \times 80 \times 80 \times 80$ tensor with a SNR of 20 dB and sample blocks of size $40 \times 40 \times 40 \times 40$. Legend: relative function value (-·-·-), Δ_k (- - -), relative step size (——), the change relative to the estimated CRB scaled by 10^{-3} (·····) and the relative CPD error E_{CPD} (——).

To convert the step restriction strategy to the full tensor, we take into account that instead of 2 iterations, now $Q = 25$ iterations are needed to update all variables. To compensate for this we take $K_{\text{search}} = 12.5 \cdot 30 \approx 400$. This results in the following strategy:

$$\Delta_k = \begin{cases} 0.8 & \text{if } k < 400 \\ 0.01 \cdot (0.85)^{(k-400)/25} & \text{if } k \geq 400 \end{cases}.$$

The result for the full tensor is shown in Figure 7. The convergence profile is indeed very similar to the $80 \times 80 \times 80 \times 80$ case. The total decomposition time for this tensor is 308 s: the decomposition of the small sample to determine the step restriction schedule takes 24 s and the decomposition of the full tensor 284 s. Note that the shuffling operation is important here. If the first index set $\mathcal{I}_1$ is not shuffled every Q_1 iterations, the rank-1 blocks will cause the algorithm to fail.

C. Influence of the block size

Another important parameter is the size of the sampled blocks. In this section, the influence of the block size on the time needed to converge, the number of data accesses and the accuracy is illustrated.

First the computation time and number of data accesses are investigated for a noiseless tensor of size $800 \times 800 \times 400$ (1.9 GB). The rank $R = 20$ and the original factor matrices are generated from a uniform distribution $\mathcal{U}(0, 1)$. No step

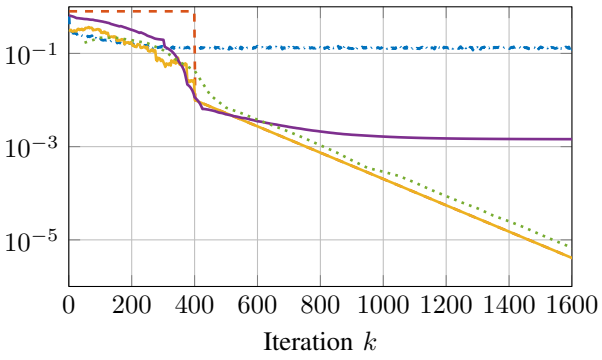

Fig. 7. Decomposition of the full $1000 \times 1000 \times 1000 \times 1000$ tensor with a SNR of 20 dB and sample blocks of size $40 \times 40 \times 40 \times 40$. Legend: relative function value (-·-·-), Δ_k (- - -), relative step size (——), the change relative to the estimated CRB scaled by 10^{-5} (·····) and the relative CPD error E_{CPD} (——).

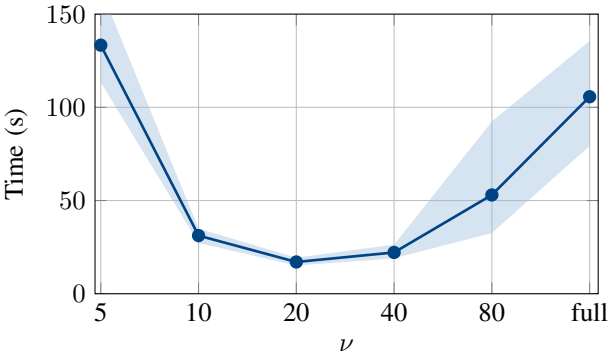

Fig. 8. Median computation time over 50 Monte Carlo experiments for blocks of size $[4 \quad 4 \quad 2] \cdot \nu$ to obtain a CPD error $E_{\text{CPD}} < 10^{-8}$. The noiseless tensor has size $800 \times 800 \times 400$ and is generated using uniformly distributed factor matrices.

size restriction schedule is needed in this noiseless exact case. The parameter ν is used to vary the sample size $[4 \quad 4 \quad 2] \cdot \nu$. The `cpd_nls` method [16], [55] is also applied to the full tensor for comparison. The algorithm is stopped when the CPD error $E_{\text{CPD}} < 10^{-8}$. Figure 8 shows timing results for this experiment. As shown in Figure 9, the computation cost per

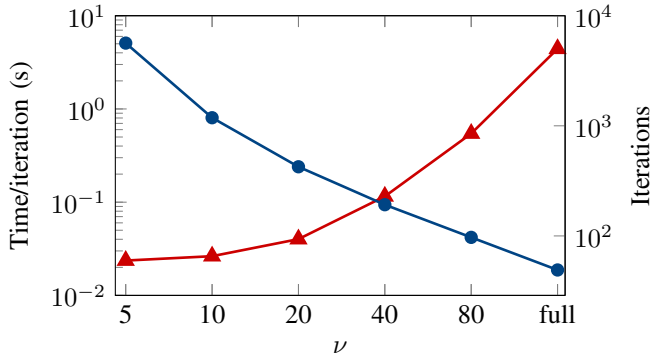

Fig. 9. Median time per iteration (—▲—, left axis) and median number of iterations (—●—, right axis) over 50 Monte Carlo experiments for blocks of size $[4 \quad 4 \quad 2] \cdot \nu$.

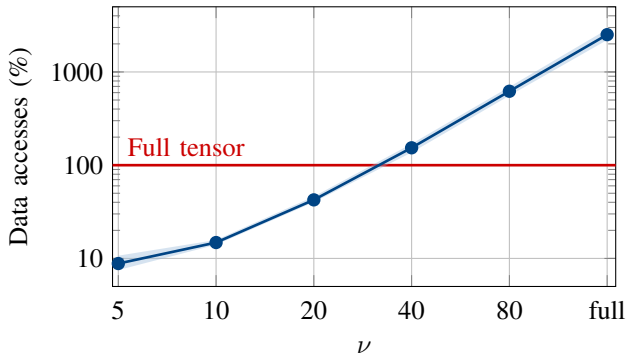

Fig. 10. Median number of data accesses over 50 Monte Carlo experiments compared to the number of entries in the tensor for blocks of size $[4 \quad 4 \quad 2] \cdot \nu$. The noiseless tensor has size $800 \times 800 \times 400$ and is generated using uniformly distributed factor matrices. The algorithm is stopped when $E_{\text{CPD}} < 10^{-8}$.

block is low when the block size is small, but many iterations are needed in order to converge. The reverse is true for large blocks. In this case, this leads to an optimal block size for $\nu = 20$ (Figure 8). However, if generating data is expensive, smaller blocks sizes may be preferable as the number of data accesses is lower, as can be seen in Figure 10. For small blocks, the algorithm finds the exact solution without accessing all elements in the tensor.

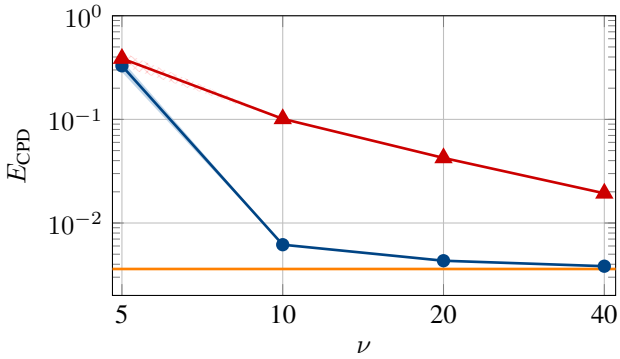

Fig. 11. Median accuracy over 50 Monte Carlo experiment for varying block sizes $[4 \quad 4 \quad 2] \cdot \nu$ without (—▲—) and with (—●—) step restriction strategy. The accuracy for the full tensor is also given (——). The rank 20 tensor of size $800 \times 800 \times 400$ is constructed using uniformly distributed factor matrices and the SNR is 20 dB. The NLS variant is used.

In the next experiment, Gaussian i.i.d. noise is added to the tensor such that the SNR is 20 dB. All other parameters are the same as in the previous experiment, except that the Cramér–Rao bound stopping criterion is now used with $\gamma = 0.01$. The resulting CPD errors are shown in Figure 11. For small block sizes, the algorithm does not find an accurate solution. Above a critical block size, the accuracy can be improved by using larger blocks. When using the step size restriction schedule the improvement no longer seems proportional as the accuracy curve flattens. Again we see that restricting the step size improves the accuracy significantly if the block sizes are large enough.

D. Classifying hazardous gasses

We conclude the experiments with a chemo-sensing application. The goal is to predict which chemical analyte, in casu a hazardous gas, is detected by an array of sensors using time series. The dataset consists of 10 different analytes, measured at six positions in a wind tunnel under different (turbulent) wind conditions and temperatures. In total 18000 time series of 260 seconds sampled at 100 Hz were collected for each of the 72 sensors [56].

Here, we only consider the measurements for CO, acetaldehyde and ammonia at the first position (L1). We perform minimal preprocessing: missing values are linearly interpolated, the 50 first and 83 last samples of each time series are trimmed, each time series is filtered using a simple moving average of length 10 and each time series is centered by subtracting its mean. One experiment is dropped because it contained too many missing values. Here, we only use spatiotemporal patterns to classify the analytes and ignore scale, wind and temperature information. Therefore, we normalize all time series for each experiment by dividing each time-sensor slice by its Frobenius norm. The resulting tensor has dimensions $25900 \times 72 \times 899$. Loading this 12.5 GB tensor into RAM takes about 10 minutes from an SSD. This is close to the largest tensor we can load on a system with 16 GB of RAM. If the tensor does not fit into RAM, the data can be split across multiple files. Each time a block is needed, the appropriate files need to be read from disk resulting in a high IO cost.

To classify the analytes, we cluster the coefficients of the spatiotemporal features, i.e. time-sensor patterns. The patterns are determined using a rank $R = 5$ CPD. Each row in the experiment mode factor matrix hence contains five coefficients, one for each of the patterns. To compute this CPD, the NLS variant with blocks of size $100 \times 36 \times 100$ is used. We do two experiments. In the first experiment no step size restriction is imposed by setting $\delta_0 = 1.5$ for all iterations. In the second experiment the step size is restricted after 2000 iterations using $1.5 \cdot 0.93^{(k-2000)}$. As initialization we used factor matrices drawn from a normal distribution. The resulting factor matrices for the second experiment are shown in Figure 12. As the experiments are grouped per analyte, we see that the top three factor vectors in the experiment mode can distinguish between the three different analytes that we consider. The noise on these vectors is due to the different wind and temperature conditions, as well as optimization noise because the variance has not been fully reduced. The distinctive coefficients are clustered using $k$means, which is repeated 100 times after which we use the most frequently occurring cluster for each experiment as the predicted cluster. The performance for both experiments can be seen in Table II. Imposing the step size restriction clearly improves the prediction accuracy, at the cost of a longer computation time. The time needed with restriction is still less than 3 minutes, however.

Using only three spatiotemporal features, a good classification accuracy can be achieved. The task will become more difficult if more analytes are used. In that case other features such as scale and wind and temperature measurements can be used to complement the pattern coefficients.

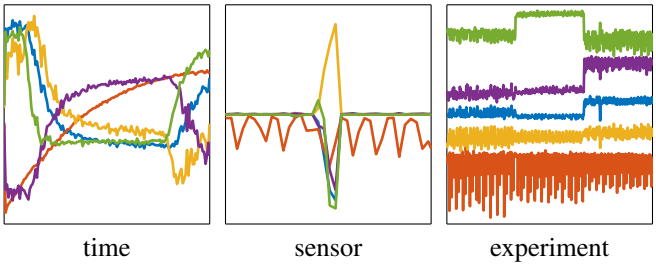

Fig. 12. Recovered factor matrices for a rank $R = 5$ CPD of the chemical analytes dataset. The vectors in the experiment mode have been shifted vertically for illustrative reasons. The top three vectors in the experiment mode appear the most distinctive and are used for classification.

TABLE II
PERFORMANCE ON THE CHEMICAL ANALYTES DATASET WITHOUT AND WITH STEP RESTRICTION. STEP RESTRICTION CLEARLY IMPROVES THE PREDICTION ACCURACY, THE PRICE BEING A SOMEWHAT LONGER COMPUTATION TIME.

	Iterations	Time (s)	Error (%)
No restriction	3000	60	5.0
Restriction	9000	170	0.3–0.8

VI. CONCLUSION

The randomized block sampling CPD algorithm presented here enables the decomposition of large-scale tensors using only small random blocks from the tensor. The advantage of using smaller blocks is twofold: small blocks can be handled more efficiently than the full tensor, and the number of data accesses needed to converge is far lower as the algorithm often needs only a fraction of all entries. We have developed two variants of our method. The ALS version is fast but needs many iterations, and thus data accesses, and may not converge for ill-conditioned datasets. The NLS variant on the other hand is robust and needs only very little data, but may be slower. We have shown that a good choice for the step size restriction schedule and block size allows our algorithm to find the CPD with an accuracy close to the accuracy achieved by state-of-the-art algorithms using the full tensor. Finally, we have introduced a new stopping criterion based on the Cramér–Rao bound that compares the actual change in variables to the expected uncertainty on the solution.

REFERENCES

[1] A. Cichocki, D. Mandic, C. Caiafa, A.-H. Phan, G. Zhou, Q. Zhao, and L. De Lathauwer, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Process. Mag.*, vol. 32, no. 2, pp. 145–163, March 2015.

[2] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[3] I. Domanov and L. De Lathauwer, "On the uniqueness of the canonical polyadic decomposition of third-order tensors—Part II: Uniqueness of the overall decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 34, no. 3, pp. 876–903, 2013.

[4] J. Kruskal, "Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics," *Linear Algebra and its Applications*, vol. 18, no. 2, pp. 95 – 138, 1977.

[5] N. Sidiropoulos and R. Bro, "On the uniqueness of multilinear decomposition of N-way arrays," *Journal of Chemometrics*, vol. 14, no. 3, pp. 229–239, 2000.

[6] A. Smilde, R. Bro, P. Geladi, and J. Wiley, *Multi-way analysis with applications in the chemical sciences*. Wiley Chichester, UK, 2004.

[7] R. Bro, "PARAFAC. Tutorial and applications," *Chemometrics and intelligent laboratory systems*, vol. 38, no. 2, pp. 149–171, 1997.

[8] N. Sidiropoulos, R. Bro, and G. Giannakis, "Parallel factor analysis in sensor array processing," *IEEE Trans. Signal Process.*, vol. 48, no. 8, pp. 2377–2388, 2000.

[9] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent component analysis and applications*. Academic press, 2010.

[10] I. Domanov and L. De Lathauwer, "Canonical polyadic decomposition of third-order tensors: Reduction to generalized eigenvalue decomposition," *SIAM J. Matrix Anal. Appl.*, vol. 35, no. 2, pp. 636–660, 2014.

[11] L. De Lathauwer, "A link between the canonical decomposition in multilinear algebra and simultaneous matrix diagonalization," *SIAM J. Matrix Anal. Appl.*, vol. 28, no. 3, pp. 642–666, 2006.

[12] E. Sanchez and B. R. Kowalski, "Generalized rank annihilation factor analysis," *Analytical Chemistry*, vol. 58, no. 2, pp. 496–499, 1986.

[13] R. Harshman, "Foundations of the parafac procedure: models and conditions for an "explanatory" multimodal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, pp. 1–84, 1970.

[14] G. Tomasi and R. Bro, "A comparison of algorithms for fitting the parafac model," *Computational Statistics & Data Analysis*, vol. 50, no. 7, pp. 1700 – 1734, 2006.

[15] E. Acar, D. Dunlavy, and T. Kolda, "A scalable optimization approach for fitting canonical tensor decompositions," *Journal of Chemometrics*, vol. 25, no. 2, pp. 67–86, 2011.

[16] L. Sorber, M. Van Barel, and L. De Lathauwer, "Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-$(L_r, L_r, 1)$ terms, and a new generalization," *SIAM J. Optim.*, vol. 23, no. 2, pp. 695–720, 2013.

[17] A.-H. Phan, P. Tichavský, and A. Cichocki, "Low complexity damped Gauss–Newton algorithms for CANDECOMP/PARAFAC," *SIAM J. Appl. Math.*, vol. 34, no. 1, pp. 126–147, 2013.

[18] P. Paatero, "A weighted non-negative least squares algorithm for three-way parafac factor analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 38, no. 2, pp. 223–242, 1997.

[19] G. Tomasi and R. Bro, "PARAFAC and missing values," *Chemometrics and Intelligent Laboratory Systems*, vol. 75, no. 2, pp. 163–180, 2005.

[20] V. de Silva and L.-H. Lim, "Tensor rank and the ill-posedness of the best low-rank approximation problem," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 3, pp. 1084–1127, 2008.

[21] N. Vervliet, O. Debals, L. Sorber, and L. De Lathauwer, "Breaking the curse of dimensionality using decompositions of incomplete tensors: Tensor-based scientific computing in big data analysis," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 71–79, Sept. 2014.

[22] W. Hackbusch, *Tensor spaces and numerical tensor calculus*, ser. Springer series in computational mathematics. Heidelberg: Springer, 2012, vol. 42.

[23] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *ArXiv e-prints*, Feb. 2013.

[24] B. Khoromskij, "Tensors-structured numerical methods in scientific computing: Survey on recent advances," *Chemometrics and Intelligent Laboratory Systems*, vol. 110, no. 1, pp. 1–19, 2012.

[25] P. Tichavský, A.-H. Phan, and Z. Koldovský, "Cramér–Rao-induced bounds for CANDECOMP/PARAFAC tensor decomposition," *IEEE Trans. Signal Process.*, vol. 61, no. 8, pp. 1986–1997, Apr. 2013.

[26] L. Sorber, M. Van Barel, and L. De Lathauwer, "Structured data fusion," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 586–600, June 2015.

[27] X. T. Vu, S. Maire, C. Chaux, and N. Thirion-Moreau, "A new stochastic optimization algorithm to decompose large nonnegative tensors," *IEEE Signal Process. Lett.*, vol. 22, no. 10, pp. 1713–1717, Oct. 2015.

[28] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, "GigaTensor: scaling tensor analysis up by 100 times-algorithms and discoveries," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 316–324.

[29] E. Papalexakis, C. Faloutsos, and N. Sidiropoulos, "ParCube: Sparse parallelizable tensor decompositions," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, P. Flach, T. De Bie, and N. Cristianini, Eds. Springer Berlin Heidelberg, 2012, vol. 7523, pp. 521–536.

[30] B. Bader and T. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM J. Sci. Comput.*, vol. 30, no. 1, pp. 205–231, 2007.

[31] N. Sidiropoulos, E. Papalexakis, and C. Faloutsos, "Parallel randomly compressed cubes: A scalable distributed architecture for big tensor

decomposition," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 57–70, 2014.
[32] J. Cohen, R. Farias, and P. Comon, "Fast decomposition of large nonnegative tensors," *IEEE Signal Process. Lett.*, vol. 22, no. 7, pp. 862–866, July 2015.
[33] A.-H. Phan and A. Cichocki, "PARAFAC algorithms for large-scale problems," *Neurocomputing*, vol. 74, no. 11, pp. 1970–1984, 2011.
[34] A. Liavas and N. Sidiropoulos, "Parallel algorithms for constrained tensor factorization via the alternating direction method of multipliers," *IEEE Trans. Signal Process.*, vol. PP, no. 99, pp. 1–1, 2015.
[35] M. Mahoney, "Randomized algorithms for matrices and data," *Foundations and Trends in Machine Learning*, vol. 3, no. 2, pp. 123–224, 2010.
[36] V. Cevher, S. Becker, and M. Schmidt, "Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics," *IEEE Signal Process. Mag.*, vol. 31, no. 5, pp. 32–43, Sept 2014.
[37] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, no. 3, pp. 400–407, Sept. 1951.
[38] B. Widrow and S. Stearns, *Adaptive signal processing*, 1st ed. Englewood Cliffs, NJ: Prentice Hall, 1985.
[39] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Large-scale matrix factorization with distributed stochastic gradient descent," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 69–77.
[40] F. Li, B. Wu, L. Xu, C. Shi, and J. Shi, "A fast distributed stochastic gradient descent algorithm for matrix factorization," in *Proceedings of the 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2014, pp. 77–87.
[41] K. Shin and U. Kang, "Distributed methods for high-dimensional and large-scale tensor factorization," in *Data Mining (ICDM), 2014 IEEE International Conference on*, Dec 2014, pp. 989–994.
[42] R. Ge, F. Huang, C. Jin, and Y. Yuan, "Escaping from saddle points — Online stochastic gradient for tensor decomposition," in *Proceedings of The 28th Conference on Learning Theory*, 2015, pp. 797–842.
[43] Y. Nesterov, "Efficiency of coordinate descent methods on huge-scale optimization problems," *SIAM J. Optim.*, vol. 22, no. 2, pp. 341–362, 2012.
[44] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Optimization for Machine Learning*, S. Sra, S. Nowozin, and S. J. Wright, Eds. MIT Press, 2011, pp. 351–368.
[45] S. Becker and Y. Le Cun, "Improving the convergence of back-propagation learning with second order methods," in *Proceedings of the 1988 connectionist models summer school*. San Matteo, CA: Morgan Kaufmann, 1988, pp. 29–37.
[46] A. Bordes, L. Bottou, and P. Gallinari, "SGD-QN: Careful quasi-newton stochastic gradient descent," *The Journal of Machine Learning Research*, vol. 10, pp. 1737–1754, 2009.
[47] N. N. Schraudolph, J. Yu, and S. Günter, "A stochastic quasi-newton method for online convex optimization," in *International Conference on Artificial Intelligence and Statistics*, 2007, pp. 436–443.
[48] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.
[49] A. Nedić and D. Bertsekas, "Convergence rate of incremental subgradient algorithms," in *Stochastic optimization: algorithms and applications*. Springer, 2001, pp. 223–264.
[50] C. Darken and J. Moody, "Towards faster stochastic gradient descent," in *NIPS*, 1991, pp. 1009–1016.
[51] L. Bottou, "Stochastic gradient tricks," in *Neural Networks, Tricks of the Trade, Reloaded*, ser. Lecture Notes in Computer Science (LNCS 7700), G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer, 2012, pp. 430–445.
[52] X. Liu and N. Sidiropoulos, "Cramér–Rao lower bounds for low-rank decomposition of multidimensional arrays," *IEEE Trans. Signal Process.*, vol. 49, no. 9, pp. 2074–2086, Sept. 2001.
[53] B.-Z. Bobrovsky, E. Mayer-Wolf, and M. Zakai, "Some classes of global Cramér–Rao bounds," *The Annals of Statistics*, vol. 15, no. 4, pp. 1421–1438, 1987.
[54] S. E. Leurgans, R. T. Ross, and R. B. Abel, "A decomposition for three-way arrays," *SIAM J. Matrix Anal. Appl.*, vol. 14, no. 4, pp. 1064–1083, 1993.
[55] L. Sorber, M. Van Barel, and L. De Lathauwer, "Tensorlab v2.02," May 2014. [Online]. Available: http://www.tensorlab.net/
[56] A. Vergara, J. Fonollosa, J. Mahiques, M. Trincavelli, N. Rulkov, and R. Huerta, "On the performance of gas sensor arrays in open sampling systems using inhibitory support vector machines," *Sensors and Actuators B: Chemical*, vol. 185, pp. 462–477, 2013.

Nico Vervliet obtained the M.Sc. degree in Mathematical Engineering from KU Leuven, Belgium, in 2013. He is a Ph.D. candidate at the STADIUS Center for Dynamical Systems, Signal Processing and Data Analytics of the Electrical Engineering Department (ESAT), KU Leuven and is affiliated with iMinds Medical IT. His research interests include decomposition algorithms for large and incomplete tensors, and the analysis of multiview data.

Lieven De Lathauwer received the Ph.D. degree from the Faculty of Engineering, KU Leuven, Belgium, in 1997. From 2000 to 2007 he was Research Associate with the Centre National de la Recherche Scientifique, France. He is currently Professor with KU Leuven. He is affiliated with the Group Science, Engineering and Technology of Kulak, with the Stadius Center for Dynamical Systems, Signal Processing and Data Analytics of the Electrical Engineering Department (ESAT) and with iMinds Medical IT. He is Associate Editor of the SIAM Journal on Matrix Analysis and Applications and has served as Associate Editor for the IEEE Transactions on Signal Processing. His research concerns the development of tensor tools for engineering applications.